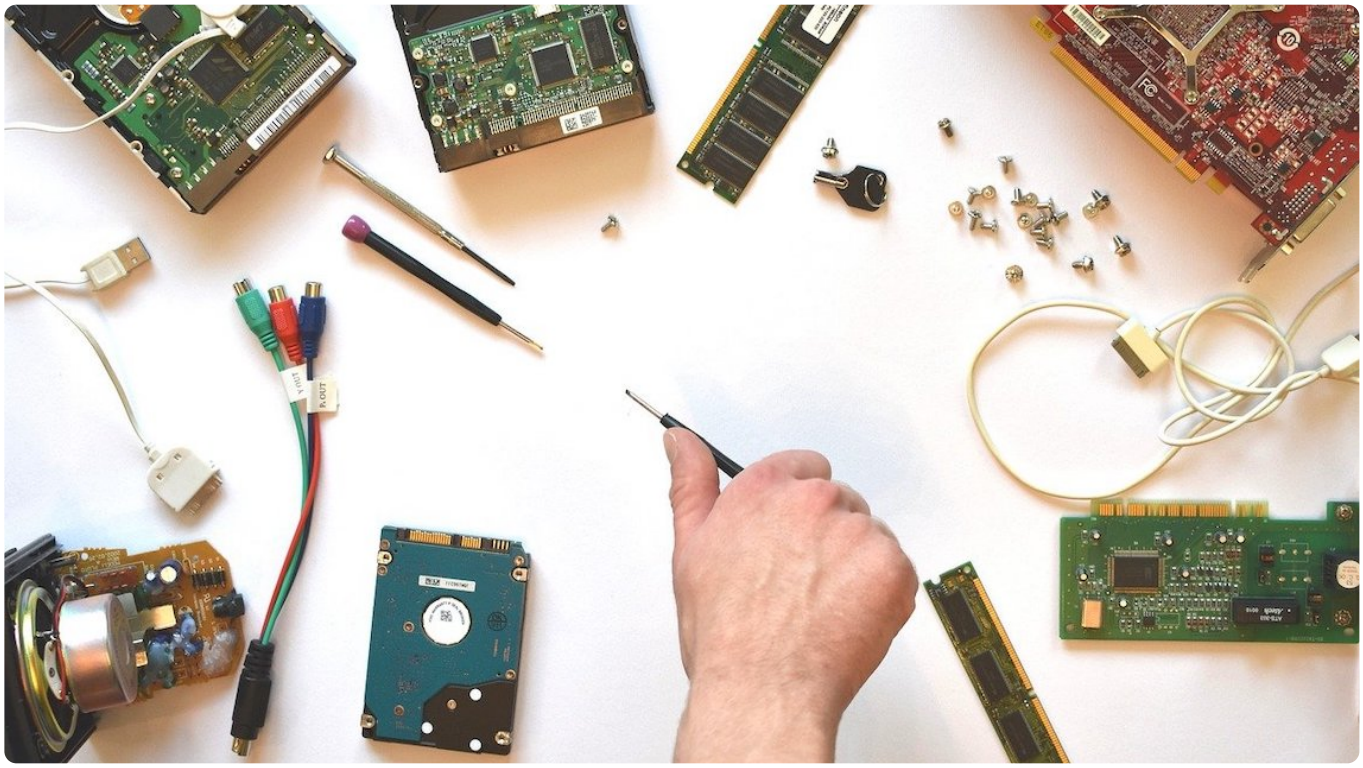
16 | 实战准备：如何搭建硬件开发环境？

2020-12-16 郭朝斌

物联网开发实战 进入课程 >



讲述：郭朝斌
时长 16:34 大小 15.19M

你好，我是郭朝斌。

从今天开始，我们就进入了课程的实战篇，我会手把手带你从 0 开始完成自己的智能家居项目。

这个项目具体包括哪些产品呢？在第 5 讲中，我们根据智能家居产品的设计原则，已经设计好了 4 个产品场景，分别是：

1. 可以手机控制的智能电灯
2. 可以基于光线自动调节的智能电灯
3. 可以语音控制的智能音箱

4. 可以基于环境温湿度和土壤湿度自动浇水的浇花器

它们分别对应了实战篇的第 17～20 讲的内容。

不过，在打造这些产品场景之前，我们还需要先搭建好**硬件开发环境**。在这一讲，我就以智能电灯为例，带你完成这个准备工作。

通信技术：Wi-Fi

为了能让手机控制电灯，我们首先要让电灯接入网络。在第 2 讲中，我们介绍了很多种通信技术，智能家居场景下常用的有 Wi-Fi、BLE 和 ZigBee 等。那么智能电灯应该选择哪种技术呢？

从**通信速率**来看，智能电灯传输的数据，包括控制命令和几种状态的数值，数据量都非常小，这几种通信技术都可以满足要求。

从**功耗**来看，智能电灯是直接连接电线的，不需要电池供电，所以低功耗的 BLE 和 ZigBee 技术不是必须的选择，功耗相对较高的 Wi-Fi 也可以考虑。

从**普及度**和**易用性**的角度分析，如果使用 BLE，设备与手机的交互确实会非常方便。但是 BLE 和 ZigBee 的设备都有一个缺点，就是需要**搭配专有的网关**才能连接互联网，这在部署和使用的时候都比较麻烦。所以，我们选择 **Wi-Fi** 作为智能电灯的通信方式。

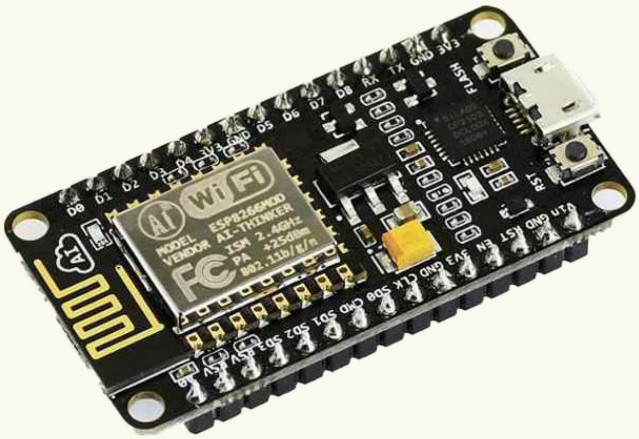
开发板：NodeMCU

确定使用 Wi-Fi 之后，我们又该怎么选开发板呢？（你可能也关心选择哪一款芯片。不过，为了方便讲解和动手实践，这里我们还是围绕开发板来展开。关于芯片的选型，我们可以另外找机会交流。）

我推荐选择开源硬件的开发板，有两个原因。第一，硬件、软件的各种技术实现是公开的，方便分析问题，也方便后期转化为量产的产品；第二，有社区氛围，使用的人比较多，大家可以针对具体的问题进行交流。

比如说 NodeMCU 就是一个不错的选择。基于 ESP8266 芯片的版本，Flash 空间有 4MB，自带 Wi-Fi 功能，而且价格便宜，在国内外都非常流行。（这里顺带说一句，

ESP8266 是国内企业的芯片产品，国内企业在 Wi-Fi 和 BLE 芯片上的优势真的是越来越明显。）



开发语言：Python

那么，开发语言用哪一种比较好呢？我计划使用 **Python**。

你可能会觉得奇怪：嗯？为什么不用 **C 语言**？

主要原因是，我不希望开发语言成为实战项目的障碍。先不说 C 语言本身的难度，光是它需要交叉编译的特性和不够便捷的调试方式，就已经很影响效率了。

相比之下，使用比较简单的 Python 语言，开发和调试都会非常方便。当然，选择 Python 还有别的好处，你在后面的实战过程中可以逐渐感受到。

如果你是嵌入式开发的高手，对 C 语言了然于胸，可以信手拈来，那你也可以基于我介绍的步骤，用 C 语言，甚至其他的语言来实践项目的编程（期待你的分享）。语言是一个工具，我们完全可以拿来灵活应用，实现我们的工作任务，而不应该成为一种羁绊。

当然，我也建议你不要排斥这次使用 Python 的机会。一方面，这次尝试可以拓展你的技术视野；另一方面，掌握 Python 对你写后台、做数据分析和写脚本也非常有帮助，可以在很多方面提高你的效率。

不过，你可能还是不放心：嵌入式硬件的计算资源都非常有限，在开发板上面运行 Python 代码可行吗？

这确实是一个挑战，好在 MicroPython 项目已经提供了解决方案。

MicroPython 是专门为**嵌入式系统**打造的 Python 实现。它完整实现了 Python3.4 的语言特性，部分支持 Python3.5 的特性。在标准库方面，MicroPython 实现了 Python 语言的一个子集，另外还增加了与底层硬件交互的库模块。

搭建 MicroPython 开发环境

接下来，我们就来把 MicroPython 部署到 NodeMCU 开发板上，准备好开发环境。

第一步：准备固件文件

首先，我们需要为 NodeMCU 准备好 MicroPython **固件文件**。MicroPython 官方已经为 ESP8266 芯片准备了现成的固件，省去了交叉编译的工作。否则，我们还需要在电脑上使用专门的编译软件，为 ESP8266 芯片编译 MicroPython 源代码。

MicroPython 的固件分为 2M、1M 和 512K 三个不同的版本，针对不同大小的 Flash 存储空间。我们下载最新的 **2M 稳定版本**（带 stable 的）就行，因为 NodeMCU 开发板的 Flash 空间是足够的。

第二步：安装烧录工具

然后，我们使用一根 USB 数据线，将 NodeMCU 开发板和电脑连接起来。

接着，我们在电脑终端运行下面的命令，安装用来烧录的工具 **esptool**：

```
1 pip install esptool
```

复制代码

esptool 安装完成后，你可以运行 esptool.py read_mac 命令，确认 NodeMCU 板子是否连接成功。连接成功后的屏幕显示是这样的：

```
→  /Users/rubin/study/iot/geektime  >esptool.py read_mac
esptool.py v2.8
Found 3 serial ports
Serial port /dev/cu.wchusbserial14230
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 40:f5:20:07:3b:52
Uploading stub...
Running stub...
Stub running...
MAC: 40:f5:20:07:3b:52
Hard resetting via RTS pin...
```

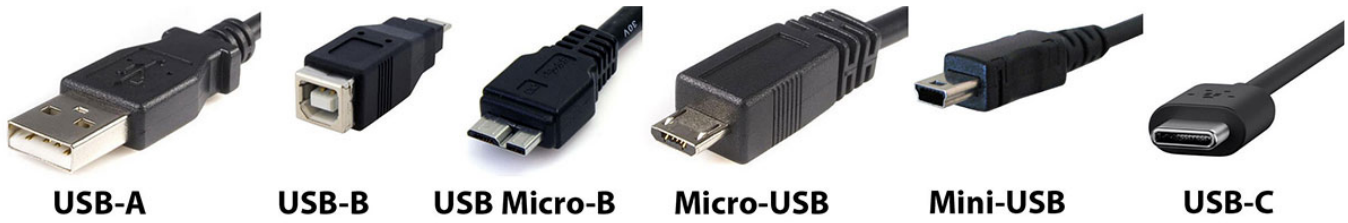
如果连接不成功，或者没有正确识别设备，屏幕上则会出现下面的结果：

```
→  /Users/rubin/study/iot/geektime  >esptool.py read_mac

esptool.py v2.8
Found 1 serial ports
Serial port /dev/cu.Bluetooth-Incoming-Port
Connecting........_____.....____.....____.....____.....____.....____...
..._____
/dev/cu.Bluetooth-Incoming-Port failed to connect: Failed to connect to Espr
essif device: Timed out waiting for packet header

A fatal error occurred: Could not connect to an Espressif device on any of t
he 1 available serial ports.
```

这时候怎么办呢？

首先，检查一下你使用的 USB 线能否传输数据。不是说笑，我就犯过这个低级错误。现在很多电子产品会随带 USB 充电线，但是为了节约成本，有些 USB 线内部实际上并没有集成两根数据信号线。你如果使用了这种线，就只能充电，而电脑是识别不出设备的。

另外，注意我们使用的数据线，一头是 USB-A 接口，另一头是 Micro-USB 接口。USB 的接口规格繁多，我在这里放了一张图，方便你区分。



不同的USB接口（图片来源：ProStorage）

如果 USB 线没有问题，那可能是电脑没有正确识别开发板，我们需要检查一下驱动文件有没有安装好。

如果你跟我一样，用的是 macOS 系统，可以在电脑的终端上输入 `ls /dev/cu*` 命令，查看是否有类似 /dev/cu.wchusbserialxxxxx 名字的设备文件。

如果你使用 Windows 系统，那么需要查看一下“设备管理器”，看看“端口（COM 和 LPT）”下面，有没有 COM* 结尾的设备。

如果没有，你可以参考这篇文章，下载相应的驱动文件安装。（注意，我的 NodeMCU 开发板使用的是 CH340 这款 USB 转串口芯片。如果是 CP2102 芯片，可以参考这篇文章。）

当你在终端看到类似下面的结果，或者在 Windows 的设备管理器中看到 COM* 设备时，就说明开发板已经成功识别。

```
→  /Users/rubin/study/iot/geektime  >ls /dev/cu*
/dev/cu.Bluetooth-Incoming-Port  /dev/cu.wchusbserial14220
```

如果仍然无法正确识别，你可以到一些论坛去交流，比如安信可的官方论坛。

第三步：烧录固件

接下来我们烧录固件。在这之前，我们需要先输入下面命令，擦除 Flash 芯片：

复制代码

```
# 注意设备名称替换为你电脑上的名称
esptool.py --port /dev/cu.wchusbserial14230 erase_flash
```

```
→  /Users/rubin/study/iot/geektime  >esptool.py --port /dev/cu.wchusbserial14230 erase_flash
esptool.py v2.8
Serial port /dev/cu.wchusbserial14230
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 40:f5:20:07:3b:52
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 9.3s
Hard resetting via RTS pin...
```

擦除成功后，我们进入存储前面下载固件的目录中，运行下面的命令，将固件文件烧录到开发板的 Flash 中：

复制代码

```
# 注意设备名称替换为你电脑上的名称，固件文件名称做类似修改
esptool.py --port /dev/cu.wchusbserial14230 --baud 460800 write_flash --flash_
```

```
→  /Users/rubin/study/iot/geektime  >esptool.py --port /dev/cu.wchusbserial14220
--baud 460800 write_flash --flash_size=detect 0 esp8266-20200911-v1.13.bin
esptool.py v2.8
Serial port /dev/cu.wchusbserial14220
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: 40:f5:20:07:3b:52
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0040
Compressed 638928 bytes to 419659...
Wrote 638928 bytes (419659 compressed) at 0x00000000 in 10.2 seconds (effective 5
02.7 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

烧录成功后，MicroPython 已经在你的开发板上运行起来了。

第四步：确认运行状态

但是开发板跟电脑不一样，是没有显示屏的，我们要怎么确认它的运行状态呢？

有一种非常简便的方法，你可以用电脑或者手机搜索一下周围的 Wi-Fi 热点，如果看到类似 “MicroPython-xxxxxx” 名称的热点（xxxxxx 是开发板 MAC 地址后半部分），说明你的 NodeMCU 工作正常。比如我的开发板 MAC 地址是“40:f5:20:07:3b:52”，现在我看到了“MicroPython-073b52”这个热点，就说明开发板在正常运行。

当然，对于 Python 来说，更方便的交互方式还是 REPL （交互式解释器），这个 MicroPython 也提供了。我们可以通过 REPL 来检验开发板的运行。

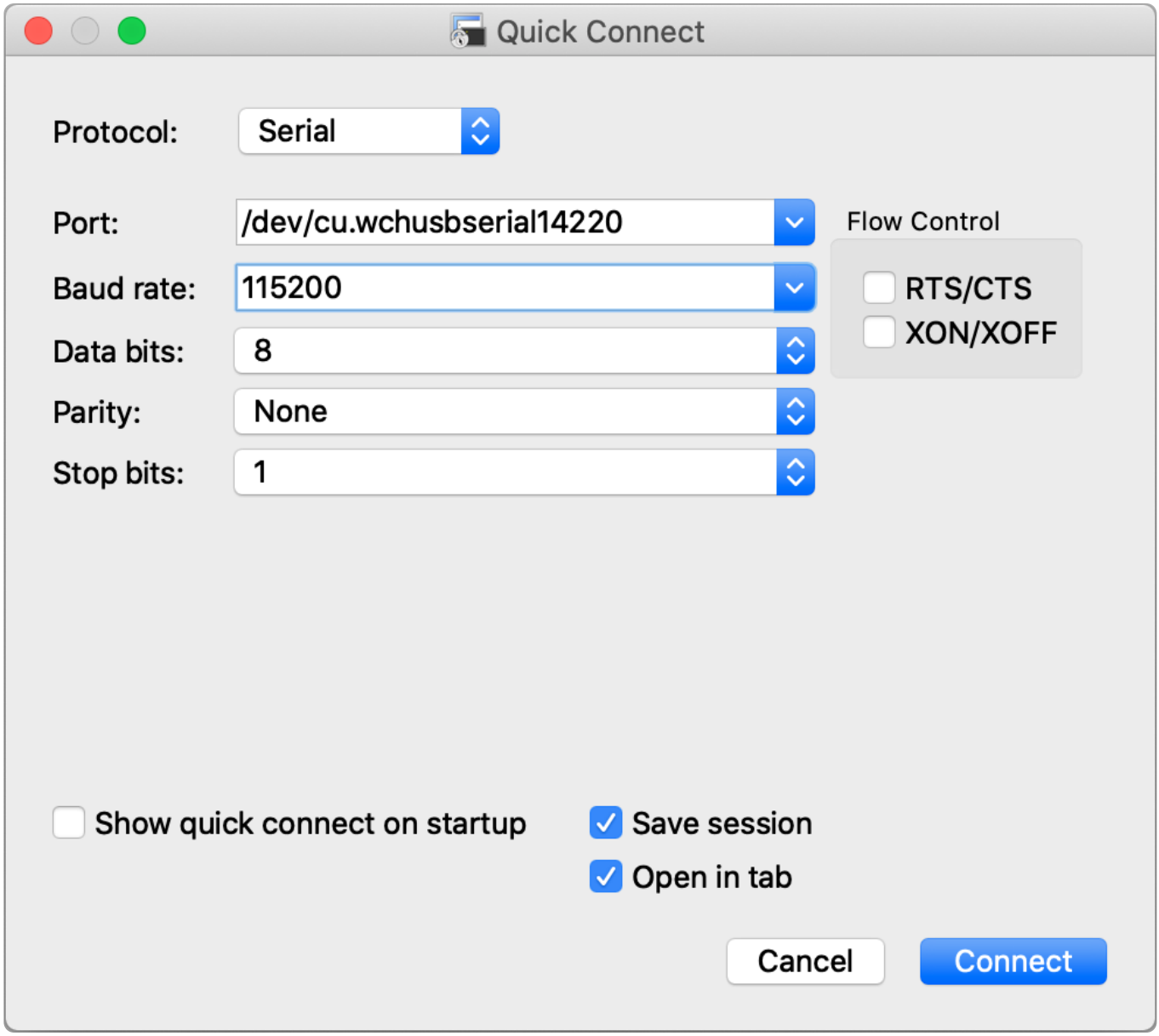
我们还是使用烧录时用到的 USB 线连接开发板和电脑。在 MacOS 电脑上，重新连接开发板的时候，串口设备名称可能会改变，所以为保险起见，再次运行命令：

```
1 ls /dev/cu*
```

复制代码

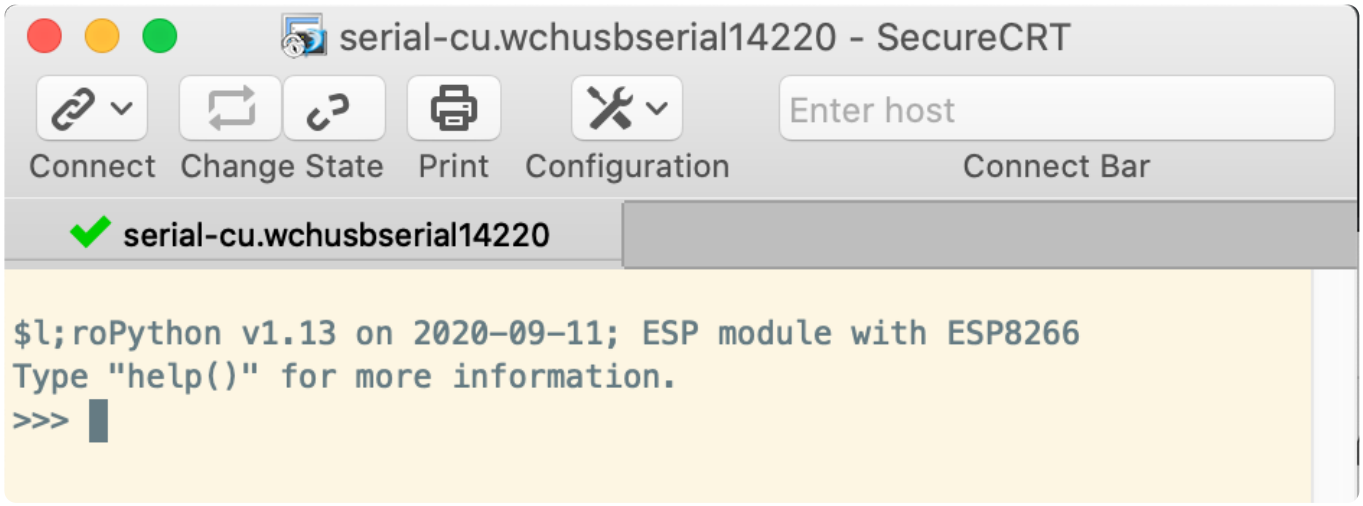
获得串口设备名称之后，我们可以使用终端模拟器软件，比如 SecureCRT，通过串口协议连接上开发板，进行交互。

需要注意的是，波特率（Baud rate）设置为 115200，这与前面烧录时选择的值不同。



如果你使用 Windows 操作系统，那么 PuTTY 更加流行。当然，建立连接的参数设置都是类似的。

成功连接后，SecureCRT 的窗口会输出类似下面的结果：



看到熟悉的符号 “>>>”，我们就知道，可以真正进行交互了。

第五步：体验交互

先用“Hello World”来个经典的打招呼吧。

接着，我们体验一下 MicroPython 控制 LED 灯。因为开发板 NodeMCU 12F 的 GPIO2 管脚接有一个 LED 灯，你可以输入下面的代码，控制它的点亮和熄灭。

复制代码

```
>>> print("Hello World from MicroPython!")
Hello World from MicroPython!
>>> import machine
>>> pin = machine.Pin(2, machine.Pin.OUT)
>>> pin.off()
>>> pin.on()
```

需要注意的是，不同的板子上，这个管脚的高低电平的设计可能不同，所以 `pin.on()` 可能是熄灭 LED 灯；`pin.off()` 反而是点亮 LED 灯。

部署代码到开发板

那么，能不能运行一个 Python 代码文件呢？比如，基于在 REPL 中尝试的点亮 LED 操作。

我们写一个代码段：

复制代码

```
import machine
import time

# 指明 GPIO2 管脚
pin = machine.Pin(2, machine.Pin.OUT)

# 循环执行
while True:
    time.sleep(2)   # 等待 2 秒
    pin.on()        # 控制 LED 状态
    time.sleep(2)   # 等待 2 秒
    pin.off()       # 切换 LED 状
```

这段代码实现的功能是，控制 LED 灯以 2 秒的间隔，不断点亮、熄灭。

为了在电路板上运行这个 Python 代码，我们需要做两件事情：

1. 将代码段保存到一个文件中，这个文件的名字必须是 main.py。
2. 将代码文件 main.py 放到开发板的文件系统中，而且是根目录。

这样，当开发板启动或者重启的时候，就会自动执行 main.py 文件中的代码。

第一点我们可以很容易做到。但是，怎么把代码文件上传到开发板上呢？

MicroPython 的官方提供了一个工具 pyboard.py，它也是基于串口连接与开发板通信的。你可以使用它操作开发板上的文件系统，比如常用的拷贝文件、创建文件夹、删除等功能，甚至可以将电脑上的代码文件加载到内存中，直接运行。这非常便于你在开发过程中，进行代码的调试。

下载 pyboard.py 的源文件到电脑后，你可以运行下面的命令，将 main.py 文件部署到你的开发板：

复制代码

```
# 设置环境变量，指明串口设备
export PYBOARD_DEVICE=/dev/cu.wchusbserial14220
```

```
3
4
5 #拷贝当前目录下的 main.py 到开发板
6 ./pyboard.py cp main.py :
```

不过，pyboard.py 在 MacOS 系统上运行有问题。比如，在电脑终端，尝试运行下面的命令，就会收到 “could not enter raw repl” 这个错误信息。

复制代码

```
1 ./pyboard.py -f ls /
```

这可能是 MacOS 上的串口芯片 CH340 的驱动的问题，它会在建立串口连接时，重置 NodeMCU ，导致 enter_raw_repl 函数无法正常执行。如果你只能在 MacOS 上开发，怎么办呢?

我试用过几种类似的工具，这里推荐你使用 AdaFruit MicroPython tool —— ampy。安装过程可以打开链接了解，我就不展开了。一般情况下，你可以用下面的命令完成安装：

复制代码

```
1 pip install adafruit-ampy
2 # ---或者---
3 pip3 install adafruit-ampy
```

ampy 是通过增加延时的方法，来规避 MacOS 系统上的问题的。所以在使用的时候，我们需要先设置一个环境变量 —— AMPY_DELAY。延时的推荐值是 0.5，不过，具体实践时，你需要根据自己的开发板的试验情况，灵活调整这个数值。

复制代码

```
1 export AMPY_DELAY=0.5
```

我们可以在终端中输入上面的指令，也可以将它加入到 .bashrc 或 .zshrc 等配置文件中，避免每次都要重复输入。

使用 ampy 的过程中，常用的环境变量还有下面两个，可以根据具体情况设置：

```
#设备名称请根据你的情况修改
export AMPY_PORT=/dev/cu.wchusbserial14220

#串口通信的波特率
export AMPY_BAUD=115200
```

然后，输入下面的命令，就把代码部署到开发板上了。

```
ampy put main.py
```

小结

在这一讲中，我带你搭建了智能电灯的硬件开发环境。

1. 在通信技术方面，我从通信速率、功耗、普及度和易用性等角度考虑，最终选择了 Wi-Fi。在实际工作中，你也可以通过同样的思路来选择其他产品的通信技术。
2. 在开发板上面，推荐选择开源硬件的开发板，因为使用和交流都更方便。实战项目选择的是 NodeMCU ESP8266，它在 Flash 空间、通信技术和价格方面有一定优势。
3. 为了降低动手实践的难度，我们选择了 Python 开发语言。而 MicroPython 为我们提供了在 NodeMCU 上使用 Python 语言的条件。
4. 在搭建 MicroPython 开发环境的过程中，我们需要使用 esptool 工具。通过 USB 线连接 NodeMCU，你可以将固件烧录到开发板的 Flash 中。

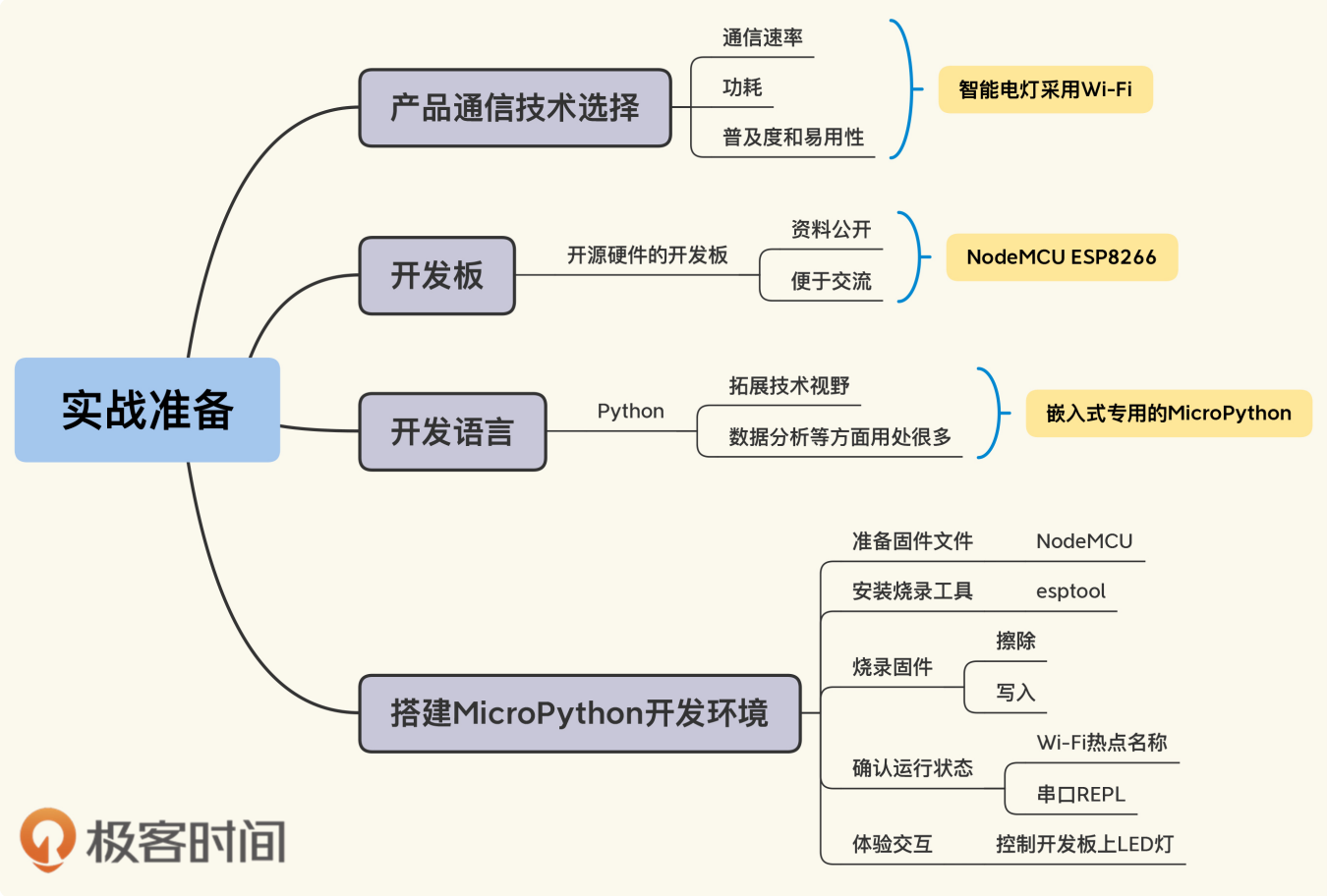
这个选择思路和搭建过程不仅适用于智能电灯，也适用于自动浇花器，它们的开发环境是一样的。至于智能音箱，使用的开发板是树莓派，我在第 15 讲已经介绍过使用方法了，所以你也不需要担心。

思考题

这一讲是实战篇的第一讲，所以最后我想给你留一个需要动手的问题。

MicroPython 是专门为嵌入式开发设计、实现的 Python 语言开发环境。在这一讲中，我们通过 GPIO 的**输出**，实现了 LED 灯的控制。你能否实现一个 GPIO 的**输入**功能，并且通过这个输入信号，控制 LED 灯的点亮和熄灭？

希望你能留言区和我交流，也欢迎你将本讲分享给你的朋友一起学习讨论。



提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 15 | 平台：智能家居开源平台的生态是怎样的？

下一篇 17 | 远程控制：怎样打造联网的智能电灯？

精选留言 (8)

写留言

Geek_eb46b7
2020-12-19

win下的记录：

esptool --port COM3 erase_flash

esptool --port COM3 --baud 460800 write_flash --flash_size=detect -fm dio 0 es...

展开

3

贤伟
2020-12-16

cp2102 driver的下载地址： https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers
（文中的链接貌似打不开了）

展开

1

SapereAude
2020-12-16

老师您好，买的时候需要带CP2012芯片模块的吗?

展开

1 1

小胖
2020-12-19

ampy --port COM3 --baud 115200 --delay 0.5 put main.py ampy把文件放到ROOT目录
ampy --port COM3 --baud 115200 --delay 0.5 ls 浏览跟目录

小胖
2020-12-19

windows 测试了下，循环点亮，命令行为 pyboard.py main.py :

Socrakit
2020-12-18

用 adafruit-ampy 之类的上传文件到开发板前要把 SecureCRT 断开，不然断开会占用，这个低级错误花了我几十分钟时间....

展开

惹忧草

2020-12-16

老师，我这烧录完成没有错误产生，但是wifi搜不到，连接串口显示
Connecting to COM3...
Connected.
键盘无响应，重复擦除烧录了3次还是不行。这个会是什么原因呢？ 该怎么排查

展开

惹忧草

2020-12-16

老师，我这烧录完成没有错误产生，但是wifi搜不到，连接串口显示Connecting to COM3...
Connected.

展开

3